

Resolution And Datalog Rewriting Under Value Invention And Equality Constraints

Bruno Marnette
marnette.bruno@gmail.com

Disclaimer: This preliminary report was originally written at the INRIA Saclay in 2010.
The list of references may therefore be incomplete at the time of this release (Dec 2012)

ABSTRACT

While Datalog is a golden standard for denotational query answering, it does not support value invention or equality constraints. The *Datalog[±]* framework introduced by Gottlob faces these issues by considering rules with fresh variables in the head (known as *tgds*) or equalities in the head (known as *egds*). Several tractable classes have been identified, among which: (S) the class of *sticky* tgds; (T) the class of tgds and egds ensuring oblivious termination; and (G) and the class of *guarded* tgds. In turn, the tractability of these classes typically relies on the ‘chase’: (S) ensures that every chase derivation is ‘sticky’; (T) ensures polynomial chase termination; and (G) allows stopping the chase after a fixed ‘depth’ while preserving completeness.

This paper shows that there are alternative algorithms (instead of the chase) that can serve as a basis for the design of (larger) tractable classes. As a first contribution, we present an algorithm for *resolution* which is complete for any set of tgds and egds (rather than being complete only for specific subclasses). We then show that a technique of saturation can be used to achieve completeness with respect to First-Order (FO) query rewriting. As an application, we generalize a few existing classes (including (S)) that ensure the existence of a finite FO-rewriting.

We then consider a more general notion of rewriting, called *Datalog rewriting*, and show that it provides a truly unifying paradigm of tractability for the family of *Datalog[±]* languages. While the classes (S), (T) and (G) are incomparable, we show that every set of rules in (S), (T) or (G) can be rewritten into an equivalent set of standard Datalog rules. On the negative side, this means intuitively that *Datalog[±]* does *not* extend the expressive power of Datalog in the context of query answering. On the positive side however, one may use the flexible syntax of *Datalog[±]* while using (only) standard Datalog in the background, thus making use of existing optimization techniques, such as Magic-Set.

1. INTRODUCTION

While the language Datalog and its extensions have been studied for decades (see e.g. [12, 2, 1]), they recently received a renewed attention. In particular Gottlob et al introduced a comprehensive and unifying framework called *Datalog[±]* ([10, 8, 9, 11]) which is based on a family of Datalog extensions. One of the main qualities of this framework lies in its generality. In particular, *Datalog[±]* is expressive

enough to cover some interesting classes of ontologies, some light-weight description logics and some fragments of F-logic (see e.g. [10]). It was also argued in [10] that *Datalog[±]* is useful in a variety of contexts, for example Data Exchange [15] and the Semantic Web [7]. The main reason for this, is that *Datalog[±]*, unlike standard Datalog, addresses the fundamental problem of *value invention* by considering rules with fresh variables in the head. Such rules are known as *tuple-generating dependencies* (*tgds* in short) and correspond to first-order formulas of the form

$$\forall \bar{x}, \bar{y}, \phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}, \psi(\bar{x}, \bar{z})$$

where ϕ and ψ are two conjunctions of atoms (that may express *joins*) while \bar{z} is a tuple of existentially quantified variables that can be used to reason about unknown entities. In addition, *Datalog[±]* supports *equality-generating dependencies* (*egds* in short) of the form

$$\forall x, y, \bar{z}, \phi(x, y, \bar{z}) \rightarrow x = y$$

where ϕ is a conjunction of atoms. Tgds and egds can be used to encode typical schema dependencies such as *inclusion dependencies* or *function dependencies* (see e.g. [1]) which, in turn, allow to reason about structured data. On the negative side, however, the problem of query answering under arbitrary tgds and egds is undecidable [6], and it was observed in [8] that the problem remains undecidable even for a *fixed* set tgds.

To avoid undecidability, the *Datalog[±]* framework typically considers (in [10]) three alternative restrictions called *termination*, *guardedness* and *stickiness*, defined as follows:

Termination: There exists a chase procedure [15, 14, 8, 20] that, for a given database, computes a universal solution in polynomial time (data complexity) which, in turn, can be used for sound and complete query answering.

Guardedness: The set of dependencies consists of *guarded* tgds and *separable* egds, in which case, as shown in [8], it is possible to reach tractability, while remaining complete, by stopping the oblivious chase after a fixed depth.

Stickiness: The set of dependencies is a set of tgds (and separable egds) ensuring that every chase derivation is *sticky* [11], meaning intuitively that the fresh variables introduced by the chase are only propagated in a harmless way.

The three classes discussed above are unfortunately incomparable, and the only property that really unifies them in [10] is the fact that they are all based on the *chase*, either in their definition or in terms of properties. This approach has several advantages and it contributes, in particular, to the simplification of the ‘big picture’. However, there are alternative ways of approaching the above classes. For example, the guarded fragment can be decided, at least in some cases, using *tableaux* algorithms or *resolution* algorithms (see e.g. [24, 19, 17, 16]). Similarly, the criterion of stickiness can be understood as a criterion ensuring the termination of resolution (as opposed to a specific property of the chase). In fact, a custom resolution procedure was proposed in [11] for the case of stickiness. However, this procedure relies on the specific properties of sticky tgds, and it is relevant only for this class of dependencies. In contrast with [11], a contribution of this paper will be to consider a resolution algorithm which is defined (and complete) for arbitrary sets of tgds and egds, as to identify larger tractable classes.

While exploring some alternatives to the chase procedure, this paper follows a similar methodology to that of Datalog[±] in the sense that it aims at identifying a *unifying* paradigm. Towards this goal, we will consider, beside resolution, several notions of *rewriting*:

- **Data Rewriting** (*a.k.a. Universal Solutions*).
Given a database \mathcal{D} and a set of dependencies Σ , a *data rewriting* for \mathcal{D} is a new database \mathcal{D}_Σ which integrates all the information can be inferred from Σ . Given such a rewriting \mathcal{D}_Σ , we can then test whether a conjunctive query Q is implied by \mathcal{D} and Σ by testing whether Q is implied by \mathcal{D}_Σ . This technique can be used whenever the chase terminates since a *universal solution* is in fact a special case of data rewriting.
- **Query Rewriting** (*a.k.a. First-Order Rewriting*).
Given a query Q and a set of dependencies Σ , a *query rewriting* for (Σ, Q) is a query Q_Σ (in a first-order language) which integrates all the information encoded in Σ . Given such a rewriting Q_Σ , we can test whether Q is implied by Σ and a database \mathcal{D} by testing whether Q_Σ is implied by \mathcal{D} . As already observed in [11], the technique of query rewriting can be used whenever Σ is a set of sticky tgds.
- **Datalog Rewriting** (*The Unifying Paradigm*).
Given a conjunctive query Q and a set of dependencies Σ , a Datalog rewriting for (Σ, Q) consists of a new pair (Σ', Q') where Σ' is a set of standard Datalog rules (without fresh variables in the head) and Q' is a query such that (Σ', Q') is equivalent to (Σ, Q) with respect to query answering.

As we will show in this paper, the technique of Datalog rewriting is not only a strictly generalisation of (first-order) query rewriting, but it can also be used in the case of terminating dependencies (instead of data rewriting) and in the case of guarded dependencies (instead of relying on the chase). In this sense, Datalog rewriting is therefore a truly unifying paradigm for the family of Datalog[±] languages. Also, Datalog rewriting was proved useful for classes of dependencies that are not (yet) covered by the Datalog[±]

framework, in particular in the context of Description Logic (see e.g. [25]). In this sense, heuristics for Datalog rewriting can be used to further generalize the Datalog[±] framework.

Structure and Main Contributions

The preliminary section formalizes the problem of query answering (under constraints) as a basic implication problem. For the sake of concision and symmetry, the key notions of *databases*, *queries* and *dependencies* are all based of sets of atoms with variables only (called *instances*). Constants, null values, and non-boolean queries with equality atoms are discussed in Section 3.5.

(1) In Section 3.1, we propose a concise definition of *resolution* for arbitrary tgds and egds and show that it is complete for query answering. In contrast with alternative approaches (such as [23]), this definition does *not* rely on skolemization and unskolemization. Instead, it relies simply on instances and renamings (a.k.a. homomorphisms).

(2) In Section 3.2, we show that a technique of saturation can be used to reach completeness with respect to finite rewritability. More precisely, the saturated chase (like the core chase of [14]) computes a finite data rewriting whenever there exists one. Similarly, the saturated resolution computes a finite query rewriting whenever there is one.

(3) In Section 3.3, we focus on tgds and revisit the notion of stickiness. We observe that there is a reduction from the class of sticky tgds to the class of lossless tgds which, in fact, can be used to identify larger classes or rewritable queries.

(4) In Section 3.4, we show that query rewriting under egds (as opposed to tgds only) is more complex. We then propose a heuristic for the integration of a set of egds into a set of tgds, thus allowing (in some cases) to rely on stickiness and query rewriting despite the presence of conflicting egds.

(5) In Section 4.1, we show that data rewriting captures the class of dependencies ensuring the termination of the oblivious chase (from [20]), and as a special case, the class of weakly acyclic dependencies (from [15]).

(6) In Section 4.2, we finally show that data rewriting also captures the class of guarded tgds. This result is the most technical and arguably the most important contribution.

The proofs (or more accurately, the proof sketches) have been included to the body of the paper. Readers unfamiliar with the problem of query answering under tgds and egds are invited to consult e.g. [10] for more examples, applications and related works. Additional proof details (for some of the proofs) can be found in [21].

2. PRELIMINARIES

We chose the *infinite model* semantic. Recall however that it coincides with the *finite model* semantic under either termination ([14, 20]), guardedness ([5]), or stickiness ([11]).

Instances and Dependencies. Let \mathcal{V} be a countable set of variables and let σ be a finite set of predicates. We assume that each $R \in \sigma$ comes with a fixed and finite arity a_R . An instance I is a set of atoms $R(\bar{v})$ where $R \in \sigma$ and \bar{v} is a tuple

of variables respecting the arity of R , that is, $\bar{v} \subseteq \mathcal{V}^{a_R}$. We let \mathcal{V}_I be the set of variables occurring in an instance I . A *tg*d is a rule $B \rightarrow H$ where B and H are two finite instances called the *body* and the *head* of r , respectively. We say that a *tg*d r is of the form $B(X, Y) \rightarrow H(X, Z)$ when $X = \mathcal{V}_B \cap \mathcal{V}_H$, $Y = \mathcal{V}_B \setminus X$, and $Z = \mathcal{V}_H \setminus X$. Such a *tg*d r is called a *Datalog rule* when $Z = \emptyset$. An *egd* r is a rule $B \rightarrow x = y$ where B is a finite instance and $x, y \in \mathcal{V}_B$. An *egd* r is of the form $B(x, y, Z) \rightarrow x = y$ when $Z = \mathcal{V}_B \setminus \{x, y\}$.

Semantics. A *renaming* is a mapping from \mathcal{V} to \mathcal{V} . Given an instance I and a renaming θ , we let $I[\theta]$ be the set of atoms $R(\theta(t_1), \dots, \theta(t_n))$ where $R(t_1, \dots, t_n) \in I$. Given two instances I and J we let $I \models J$ when $J[\theta] \subseteq I$ for some renaming θ . Given an instance I and a set \mathcal{J} of instances, we let $I \models \mathcal{J}$ when $I \models J$ for some $J \in \mathcal{J}$. Given two sets \mathcal{I} and \mathcal{J} of instances, we let $\mathcal{I} \models \mathcal{J}$ when $I \models J$ for all $I \in \mathcal{I}$. Note that these definitions are consistent with respect to singletons. In particular, we have $I \models J$ iff $\{I\} \models \{J\}$. Given two (sets of) instances \mathcal{I} and \mathcal{J} we write $\mathcal{I} \equiv \mathcal{J}$ when both $\mathcal{I} \models \mathcal{J}$ and $\mathcal{J} \models \mathcal{I}$.

Given $X \subseteq \mathcal{V}$, a *renaming of X* is a renaming θ such that $\theta(v) = v$ for all $v \in (\mathcal{V} \setminus X)$. In the following, we use the notation θ_X to indicate that θ_X is a renaming of X . Given $Y \subseteq \mathcal{V}$ disjoint from X , and given a renaming θ_Y , we denote by $[\theta_X, \theta_Y]$ the renaming of $X \cup Y$ that coincides with θ_X on X and with θ_Y on Y .

Given an instance I and a *tg*d $r : B(X, Y) \rightarrow H(X, Z)$, we let $I \models r$ when for all θ_X and θ_Y such that $B[\theta_X, \theta_Y] \subseteq I$, there exists θ_Z such that $H[\theta_X, \theta_Z] \subseteq I$. Given an instance I and an *egd* $r : B(x, y, Z) \rightarrow x = y$, we write $I \models r$ when for all θ_x, θ_y and θ_Z such that $B[\theta_x, \theta_y, \theta_Z] \subseteq I$, it holds that $\theta_x(x) = \theta_y(y)$. Given a set Σ of *tg*ds and *eg*ds, we finally write $I \models \Sigma$ when $I \models r$ for all *tg*ds and *eg*ds $r \in \Sigma$.

Note that, for two instances I and J , the property $I \models J$ corresponds to several equivalent intuitions: the *boolean conjunctive query* J is true in the *database* I ; the query I is *contained* in the query J ; the *formula* J is *implied* by I ; the instance I is a *model* of J ; or there is an *homomorphism* from J to I . A similar comment holds for sets of instances.

In the following definition, \mathcal{D} and \mathcal{Q} denote two sets of instances and Σ denotes a set of dependencies. Intuitively: \mathcal{D} corresponds to a database or a set of databases (representing a set of possible models M), Σ corresponds to an ontology or a set of structural dependencies; and \mathcal{Q} corresponds to a union of boolean conjunctive queries.

DEFINITION 1. We say that \mathcal{Q} is *certain* in \mathcal{D} under Σ , denoted $\mathcal{D} \wedge \Sigma \models \mathcal{Q}$, iff, for all instance M such that $M \models \mathcal{D}$ and $M \models \Sigma$, we have $M \models \mathcal{Q}$.

Chase. We next recall the definition of the *chase* [15, 14, 8, 20]. More precisely, the following definition coincides with [8] in the case of *tg*ds and with [15] in the case of *eg*ds.

Given an instance I and a set of variables Z , we say that θ_Z is an *I -fresh renaming* when θ_Z is a renaming of Z , θ_Z is injective on Z and its image $\theta_Z(Z)$ is disjoint from \mathcal{V}_I . Given two variables u and v we denote by $[u \leftarrow v]$ the unique renaming θ of $\{v\}$ satisfying $\theta(v) = u$. Given an instance I , and a set Σ of *tg*ds and *eg*ds, we write $I \xrightarrow{\text{chase}}_{\Sigma} J$ when either $J = I$ or one of the following rules applies:

- (tg) There is a *tg*d $r : B(X, Y) \rightarrow H(X, Z)$ in Σ and some renamings θ_X and θ_Y such that $B[\theta_X, \theta_Y] \subseteq I$ and $J = I \cup H[\theta_X, \theta_Z]$ for some I -fresh renaming θ_Z .
- (egd) There is an *egd* $r : B(x, y, Z) \rightarrow x = y$ in Σ and some renamings θ_x, θ_y , and θ_Z such that $B[\theta_x, \theta_y, \theta_Z] \subseteq I$ and $J = I[\theta_y(y) \leftarrow \theta_x(x)]$.

Using the symbol $*$ for the transitive closure, we finally write $J \in \text{Chase}(I, \Sigma)$ when $I \xrightarrow{\text{chase}}_{\Sigma}^* J$.

3. FIRST-ORDER RESOLUTION

3.1 Definition and Completeness

We next propose a definition of *resolution* for *tg*ds and *eg*ds which, unlike alternative approaches (such as [23]), does not require any complex algorithm of (un)skolemization.

Given an instance Q and a set Σ of *tg*ds and *eg*ds, we write $Q \xrightarrow{\text{resol}}_{\Sigma} R$ when one of the following rules applies:

- (ren) $R = Q[\theta]$ for some renaming θ .
- (tg) There is a *tg*d $B(X, Y) \rightarrow H(X, Z)$ in Σ and three renamings $\theta_X, \theta_Y, \theta_Z$ such that

$$R = (Q \setminus H[\theta_X, \theta_Z]) \cup B[\theta_X, \theta_Y]$$

and the following conditions hold:

- there is at least one atom in $Q \cap H[\theta_X, \theta_Y]$;
- θ_Z is an injection from Z to $(\mathcal{V} \setminus \mathcal{V}(R))$.

- (egd) There is an *egd* $B(x, y, Z) \rightarrow x = y$ in Σ and three substitutions $\theta_x, \theta_y, \theta_Z$, such that $\theta_x(x)$ occurs several times in B , $\theta_x(x) \neq \theta_y(y)$, and R is obtained by:
 - first renaming some occurrences of $\theta_x(x)$ into $\theta_y(y)$;
 - and then adding the atoms of $B[\theta_x, \theta_y, \theta_Z]$.

We finally write $R \in \text{Resol}(Q, \Sigma)$ when $Q \xrightarrow{\text{resol}}_{\Sigma}^* R$.

THEOREM 1. For all set Σ of *tg*ds and *eg*ds, and for all instances D and Q , the following statements are equivalent:

- (1) $D \wedge \Sigma \models Q$
- (2) $\exists U \in \text{Chase}(I, \Sigma), U \models Q$
- (3) $\exists R \in \text{Resol}(Q, \Sigma), D \models R$

PROOF SKETCH. The equivalence between (1) and (2) is well-known. Recall however that it holds only under the infinite model semantics (see e.g. [14] for more details). The implication (3) \Rightarrow (1) means that the resolution is *sound*, and it is fairly straightforward. We next prove that (2) implies (3). We proceed by induction, and show that the following property holds for all $n \geq 0$:

If there is a chase derivation of length n of the form

$$D = I_0 \xrightarrow{\text{chase}}_{\Sigma} I_1 \xrightarrow{\text{chase}}_{\Sigma} \dots \xrightarrow{\text{chase}}_{\Sigma} I_n \text{ where } I_n \models Q$$

then there exists $R \in \text{Resol}(Q, \Sigma)$ such that $D \models R$.

In the case $n = 0$, the property holds with $R = Q$. Assume now the property true for $n - 1$ and consider a derivation of length n as above. Let h be a renaming such that $Q[h] \subseteq I_n$ and let $Q' = Q[h]$. Because of the resolution rule (ren), we have $Q \xrightarrow{\text{resol}}_{\Sigma} Q'$. We now distinguish two cases:

(1) Assume that I_n is obtained by chasing a tgds

$$B(X, Y) \rightarrow H(X, Z)$$

and consider $\theta_X, \theta_Y, \theta_Z$ such that $B[\theta_X, \theta_Y] \subseteq I_{n-1}$ and $I_n = I_{n-1} \cup H[\theta_X, \theta_Z]$. If $Q' \subseteq I_{n-1}$, we can apply the induction hypothesis for $n-1$. Otherwise, there is at least one atom in $Q' \cap H[\theta_X, \theta_Z]$ and we have $Q' \xrightarrow{\text{resol}}_\Sigma R$ for

$$R = (Q' \setminus H[\theta_X, \theta_Z]) \cup B[\theta_X, \theta'_Y].$$

Since $R \subseteq I_{n-1}$ we have $I_{n-1} \models R$ and we can easily conclude the inductive step.

(2) Assume that I_n is obtained by chasing an egd

$$B(x, y, Z) \rightarrow x = y$$

and consider $(\theta_x, \theta_y, \theta_Z)$ such that $B[\theta_x, \theta_y, \theta_Z] \subseteq I_{n-1}$ and $I_n = I_{n-1}[\theta']$ for $\theta' = [\theta_y(y) \leftarrow \theta_x(x)]$. For each atom $A \in Q$, choose an atom $c(A) \in I_{n-1}$ such that $\theta'(c(A)) = A$ and let $R = \{c(A), A \in Q'\} \cup B[\theta_x, \theta_y, \theta_Z]$. We can finally check that $Q' \xrightarrow{\text{resol}}_\Sigma R$ and conclude the inductive step. \square

3.2 Rewriting and Saturation

We next formalize the notions of data rewriting and query rewriting before showing that completeness (with respect to finite rewritability) can be reached, in both cases, by *saturating* the relation $\xrightarrow{\text{chase}}$ or $\xrightarrow{\text{resol}}$ in a rather natural way.

DEFINITION 2. Consider a set Σ of dependencies and two sets \mathcal{D} and \mathcal{Q} of instances. A data rewriting for (\mathcal{D}, Σ) is a set of instances \mathcal{U} such that, for all set Q' of instances:

$$(\mathcal{D} \wedge \Sigma \models Q') \text{ iff } (\mathcal{U} \models Q').$$

Similarly, a query rewriting for (Σ, \mathcal{Q}) is a set of instances \mathcal{R} such that, for all set \mathcal{D}' of instances:

$$(\mathcal{D}' \wedge \Sigma \models \mathcal{Q}) \text{ iff } (\mathcal{D}' \models \mathcal{R}).$$

Saturation. Let $\xrightarrow{\circ} \in \{\xrightarrow{\text{chase}}, \xrightarrow{\text{resol}}\}$. Given two finite sets of instances \mathcal{I} and \mathcal{J} , we write $\mathcal{I} \xrightarrow{\circ} \mathcal{J}$ when, intuitively, \mathcal{J} is a concise representation (up to equivalence) of all the instances J such that $I \xrightarrow{\circ} J$ for some $I \in \mathcal{I}$. More formally, we let $\mathcal{I} \xrightarrow{\circ} \mathcal{J}$ when \mathcal{J} can be returned by the following non-deterministic algorithm:

```

Start with  $\mathcal{J} := \mathcal{I}$ .
Repeat (until fixed point)
  If there is an instance  $J$  such that:
    (1)  $I \xrightarrow{\circ} J$  for some  $I \in \mathcal{I}$ ; and
    (2.1) we are in the case  $\xrightarrow{\circ} = \xrightarrow{\text{chase}}$  and
           there is no  $J' \in \mathcal{J}$  such that  $J \models J'$ ; or
    (2.2) we are in the case  $\xrightarrow{\circ} = \xrightarrow{\text{resol}}$  and
           there is no  $J' \in \mathcal{J}$  such that  $J' \models J$ 
  Then let  $\mathcal{J} := \mathcal{J} \cup \{J\}$ 
Return  $\mathcal{J}$ .

```

Note that the above definition can easily be translated into an actual algorithm since the set of instances I satisfying the point (1) is finite up to isomorphism (and can be enumerated in exponential time). We define a *complete derivation* for \mathcal{I} and $\xrightarrow{\circ}$ as an infinite series $(\mathcal{I}_i)_{i \geq 0}$ where $\mathcal{I}_0 = \mathcal{I}$ and $\mathcal{I}_i \xrightarrow{\circ} \mathcal{I}_{i+1}$ for each $i \geq 0$. We finally let $\mathcal{J} \in \text{Fix}(\mathcal{I}, \xrightarrow{\circ})$ when $\mathcal{J} = \bigcup_i \mathcal{I}_i$ for some complete derivation $(\mathcal{I}_i)_{i \geq 0}$.

THEOREM 2. For all set Σ of tgds and egds, and for all sets \mathcal{D} and \mathcal{Q} of instances:

- Every $\mathcal{U} \in \text{Fix}(\mathcal{D}, \xrightarrow{\text{chase}}_\Sigma)$ is a data rewriting of (\mathcal{D}, Σ) .
- Every $\mathcal{R} \in \text{Fix}(\mathcal{Q}, \xrightarrow{\text{resol}}_\Sigma)$ is a query rewriting of (Σ, \mathcal{Q}) .

PROOF SKETCH. The result would follow directly from Theorem 1 if we had removed the requirements (2.1) and (2.2) in the above definition of saturation. To prove that the result still holds with (2.1) and (2.2), it is enough to observe that the chase and the resolution are both monotone:

- If $J \models J'$ and $J' \xrightarrow{\text{chase}}_\Sigma K'$, then there exists an instance K such that $J \xrightarrow{\text{chase}}_\Sigma^* K$ and $K \models K'$.
- If $J' \models J$ and $J' \xrightarrow{\text{resol}}_\Sigma K'$, then there exists an instance K such that $J \xrightarrow{\text{resol}}_\Sigma^* K$ and $K' \models K$.

The monotonicity of the chase is well-known (see e.g. [20]). For the monotonicity of the resolution, consider $J' \models J$ and $J' \xrightarrow{\text{resol}}_\Sigma K'$. The case where K' is obtained from J' with the resolution rule (ren) is straightforward. Consider now the case (tgds) and unfold the definition of resolution so that

$$K' = (J' \setminus H[\theta_X, \theta_Z]) \cup B[\theta_X, \theta_Y].$$

Let h be a renaming of \mathcal{V}_J such that $J[h] \subseteq J'$. If $J[h]$ does not intersect $H[\theta_X, \theta_Z]$ we have $J[h] \subseteq K'$ and the property holds for $K = K'$. Otherwise, considering the instance

$$K = ((J[h]) \setminus H[\theta_X, \theta_Z]) \cup B[\theta_X, \theta_Y].$$

we can check that $I \xrightarrow{\text{resol}}_\Sigma I[h] \xrightarrow{\text{resol}}_\Sigma K$ and $K[h] \subseteq K'$. Consider finally the case (egd) and unfold the definition of resolution for an egd $B(x, y, Z) \rightarrow x = y$ of Σ and three substitutions $\theta_x, \theta_y, \theta_Z$. For each atom $A \in J'$, let $c(A)$ be the atom of K' that has been obtained from A by renaming some occurrences of $\theta_x(x)$ into $\theta_y(y)$, and observe that:

$$K' = \cup \{c(A) | A \in J'\} \cup B[\theta_x, \theta_y, \theta_Z].$$

Let h be a renaming of \mathcal{V}_J such that $J[h] \subseteq J'$ and let

$$K = \cup \{c(A) | A \in J[h]\} \cup B[\theta_x, \theta_y, \theta_Z].$$

As in the previous case, we can finally check that we have $I \xrightarrow{\text{resol}}_\Sigma I[h] \xrightarrow{\text{resol}}_\Sigma K$ and $K[h] \subseteq K'$. This concludes the proof of monotonicity and the proof of Theorem 2. \square

Termination. Let $(\mathcal{I}_i)_{i \geq 0}$ be a complete derivation of \mathcal{I} and $\xrightarrow{\circ}$. We say that this derivation *terminates* iff there exists a finite i such that $\mathcal{I}_{i+1} = \mathcal{I}_i$. Note that, in this case, we also have $\mathcal{I}_j = \mathcal{I}_i$ for all $j \geq i$. We finally say that $\text{Fix}(\mathcal{I}, \xrightarrow{\circ}_\Sigma)$ is finite when *all* the instances $J \in \text{Fix}(\mathcal{I}, \xrightarrow{\circ}_\Sigma)$ are finite, meaning (equivalently) that *all* the derivations of \mathcal{I} by $\xrightarrow{\circ}_\Sigma$ are finite.

THEOREM 3. Given a finite set Σ of tgds and two finite sets \mathcal{D} and \mathcal{Q} of instances, the following statements hold:

- If there exists a finite data rewriting for (\mathcal{D}, Σ) , then $\text{Fix}(\mathcal{D}, \xrightarrow{\text{chase}}_\Sigma)$ is finite.
- If there exists a finite query rewriting for (Σ, \mathcal{Q}) , then $\text{Fix}(\mathcal{Q}, \xrightarrow{\text{resol}}_\Sigma)$ is finite.

PROOF SKETCH. The two points are similar and we prove here the second one. Suppose that there exists a finite query rewriting \mathcal{R} for (Σ, \mathcal{Q}) and a consider a complete derivation $(\mathcal{I}_i)_{i \geq 0}$ for \mathcal{Q} and $\xrightarrow{\text{resol}}_{\Sigma}$. Since \mathcal{R} and $\cup_i \mathcal{I}_i$ are two rewritings of (Σ, \mathcal{Q}) , we have $\mathcal{R} \equiv \cup_i \mathcal{I}_i$. Since \mathcal{R} is finite, there exists a finite k such that $\mathcal{R} \equiv \mathcal{I}_k$. We can then observe that $\mathcal{I}_k = \mathcal{I}_{k+1}$ and conclude that $(\mathcal{I}_i)_{i \geq 0}$ is finite. \square

Note that a direct consequence of Theorems 2 and 3 is the following: as soon as there is *one* finite fixed point in $\text{Fix}(\mathcal{D}, \xrightarrow{\text{chase}}_{\Sigma})$ or $\text{Fix}(\mathcal{Q}, \xrightarrow{\text{resol}}_{\Sigma})$, it is the case that *all* the fixed points are finite. In particular, this means that all saturation strategies are equivalent with respect to termination, both in the case of $\xrightarrow{\text{chase}}_{\Sigma}$ and $\xrightarrow{\text{resol}}_{\Sigma}$.

3.3 Rewritable Classes of Tgds

This section illustrates how the resolution procedure from Section 3.1 can be used to ‘simplify’ the big picture on query rewritability. In particular, we first show that it provides a concise proof of finite query-rewritability of a few well-known classes of dependencies. In fact, the results stated in the following proposition are rather well-known. It is however interesting to compare the following proof sketch (based on resolution, and arguably simple) with, for example, the seminal paper of Johnson and Klug [18] where a proof was given (based on the chase, and arguably complex) for the tractability of *inclusion dependencies*. This class of dependencies is indeed covered (strictly) by the class of *local-as-view* tgds (*lav* tgds) defined in the following proposition.

PROPOSITION 1. *In each of the following cases, we can effectively compute a finite query rewriting for (Σ, \mathcal{Q}) :*

- (Lav tgds) Σ is a set of tgds $B \rightarrow H$ where the body B contains at most one atom.
- (Lossless tgds) Σ is a set of tgds $B \rightarrow H$ where, for each atom $A_h \in H$, we have $\mathcal{V}_B \subseteq \mathcal{V}_{A_h}$.
- (Acyclicity) Σ is a set of tgds and there is a linear order \leq_{Σ} on the predicates of Σ such that, for all tgd $B \rightarrow H$ in Σ , all predicate R_b occurring in B , and all predicate R_h occurring in H , we have $R_h \leq R_b$.

PROOF SKETCH. In the case of lav tgds, we can observe that the resolution rules (hom) and (tgd) never increase the number of atoms. More formally, whenever $I \xrightarrow{\text{resol}}_{\Sigma} J$, we have $|J| = |I|$. There are therefore a finite number of instances (up to \equiv) that can be computed by resolution. As a consequence, every (complete) derivation terminates and every $\mathcal{R} \in \text{Fix}(\mathcal{D}, \xrightarrow{\text{resol}}_{\Sigma})$ is a finite query rewriting. In the case of lossless tgds, the result follows from a very similar observation: the resolution rules (hom) and (tgd) never increase the number of variables. (That is, $I \xrightarrow{\text{resol}}_{\Sigma} J$ implies $\mathcal{V}_J \subseteq \mathcal{V}_I$). Consider finally the case of an acyclic set Σ of tgds. Let σ_{Σ} be the set of predicates occurring in Σ and consider the ordering $\sigma_{\Sigma} = \{R_1, \dots, R_n\}$ where $R_{i-1} \leq_{\Sigma} R_i$ for each $i \leq n$. For each instance I , consider $s(I) = (a_1, \dots, a_n)$ where, for each $i \in \{1, \dots, n\}$, a_i is the number of atoms $R'(\bar{v}) \in I$ where $R' = R_i$. We can observe that, whenever $I \xrightarrow{\text{resol}}_{\Sigma} J$, the tuple $s(J)$ is smaller than $s(I)$ with respect to lexicographic order. We can finally conclude as in the previous cases. \square

We next shed more light on the notion of stickiness from [11] which was discussed in the introduction. In particular, we show that there is in fact a direct reduction (preserving the property of finite rewritability) from the class of sticky tgds to the class of lossless tgds. This reduction proves useful in two ways: (1) it provides a direct proof of rewritability for the class of sticky tgds (which, unlike [11], does not require a custom resolution algorithm), (2) and it also allows us to identify a more general class of rewritable dependencies. In a nutshell, the key idea of the following reduction consists in replacing an atom $A(\bar{x}, \bar{y})$ by an atomic formula $R(\bar{x}) \approx (\exists \bar{y}, A(\bar{x}, \bar{y}))$ whenever $A(\bar{x}, \bar{y})$ is the only atom (in a given tgd) where the variables of \bar{y} occur.

Simplifying atoms. Given a tgd $r : B \rightarrow H$ and an atom A in the body B , we let $X_{A,r} = \mathcal{V}_A \cap \mathcal{V}_H$ and $Y_{A,r} = \mathcal{V}_A \setminus X_{A,r}$. We then say that A can be simplified in r when $Y_{A,r}$ is non-empty and disjoint from $\mathcal{V}_{B \setminus A}$. For example, in the tgd

$$A(x_1, y_1), B(x_1, x_2, y_2), C(x_1, y_3), D(y_2) \rightarrow R(x_1, x_2, z_1)$$

the atoms $A(x_1, y_1)$ and $C(x_1, y_3)$ can be simplified. In contrast, $B(x_1, x_2, y_2)$ cannot be simplified because y_2 occurs in another atom of the body.

Consider now a set Σ of tgds, a tgd $r : B \rightarrow H$ in Σ , and an atom A which can be simplified in r . Given a tgd $r' : B' \rightarrow H'$ in Σ' and an atom $A' \in H'$ we say that r' unifies with (A, r) when there exists a renaming θ_1 of \mathcal{V}_A and a renaming θ_2 of $\mathcal{V}_{B'} \cap \mathcal{V}_{H'}$ such that $A[\theta_1] \in H'[\theta_2]$, in which case (θ_1, θ_2) is called a *unifier* of r' and (A, r) . The following algorithm describes the new set of tgds $\Sigma_{A,r}$ that results from the simplification of (A, r) in Σ :

Let $\bar{x} = (x_1, \dots, x_n)$ be an ordering of $X_{A,r} = \{\bar{x}\}$
 Let R_a be a fresh predicate of arity n
 Replace A by $R_a(\bar{x})$ in the body of r
 For all tgd $r' : B' \rightarrow H'$ in Σ , including $r' = r$
 For all unifier (θ_1, θ_2) of r' and (A, r)
 Add the tgd $B'[\theta_2] \rightarrow R_a(\theta_1(\bar{x}))$.

Note that the quantification of the variables may be modified in this process, and new simplifications may therefore be possible in $\Sigma_{A,r}$. For instance, the set of tgds

$$\begin{aligned} r_1 : A(x, x, y, z, t) &\rightarrow B(x, y) \\ r_2 : C(x, y) &\rightarrow \exists u, v, A(x, y, u, v, v) \\ r_3 : D(x, y, z, t) &\rightarrow A(x, x, y, z, t) \end{aligned}$$

will, after a simplification step in r_1 , be replaced by

$$\begin{aligned} r'_1 : R_a(x, y) &\rightarrow B(x, y) \\ r_2 : C(x, y) &\rightarrow \exists u, v, A(x, y, u, v, v) \\ r_3 : D(x, y, z, t) &\rightarrow A(x, x, y, z, t) \\ r'_2 : C(x, x) &\rightarrow \exists u, R_a(x, u) \\ r'_3 : D(x, y, z, t) &\rightarrow R_a(x, y) \end{aligned}$$

and the first atom of r'_3 can now be simplified (even though this atom could not be simplified in r_3).

Despite the previous observation, we can check that the process of simplification always terminates since each step introduces only a finite number of tgds, and the number of variables in each of these tgds is strictly decreasing. For a finite set Σ of tgds, we finally define (non-deterministically) the set $\Sigma \downarrow$ as a set of tgds obtained from Σ by repeating the operation of simplification until a fixed point is reached.

THEOREM 4. *For all finite sets Σ of tgds and all finite sets \mathcal{Q} of instances, there is a finite query rewriting for (Σ, \mathcal{Q}) iff there is a finite query rewriting for $(\Sigma \downarrow, \mathcal{Q})$.*

PROOF SKETCH. Consider a series of simplification steps s_1, \dots, s_n where each s_i is characterized by the atom $A_i(\bar{x}_i, \bar{y}_i)$ that has been simplified at step s_i and the corresponding atom $R_i(\bar{x}_i)$ that has been introduced. For each i , consider the tgd $r_i : A_i(\bar{x}_i, \bar{y}_i) \rightarrow R_i(\bar{x}_i)$. Finally, let $\Gamma = \{r_i\}_{i \in \{1..n\}}$. We can observe that, for all instance D over the original schema, and every data rewriting D_Γ for (\mathcal{D}, Γ) , we have $D \wedge \Sigma \models \mathcal{Q}$ iff $D_\Gamma \wedge \Sigma \downarrow \models \mathcal{Q}$. Since, Γ is a set of Datalog rules, there is a data rewriting D_Γ which is finite iff D is finite. We can also observe that the set of tgds $\Sigma^{-1} = \{R_i(\bar{x}_i) \rightarrow \exists \bar{y}_i A_i(\bar{x}_i, \bar{y}_i)\}$ is acyclic, and for all instance D' of the extended schema, there is therefore a data rewriting $D'_{\Gamma^{-1}}$ of (D', Σ^{-1}) which is finite iff D' is finite. With letting Π be the operation that projects an instance of the extended schema on the original schema, we can then observe that, for all instance D of the original schema, we have $\Pi((D_\Gamma)_{\Gamma^{-1}}) \equiv D$. This means intuitively that there exists a one-to-one correspondence (which preserves finiteness) between the instances of the original schema and the instances of the extended schema. This is the key argument behind the proof of Theorem 4. \square

Stickiness (Slightly Revisited). Given a set of atoms B and a term t , we denote by $\text{pos}(t, B)$ the set of pairs (R, i) , called *positions*, such that B contains an atom $R(t_1, \dots, t_n)$ where $t_i = t$. Given a set of tgds Σ , a tgd $B \rightarrow H$ in Σ and an atom $A \in H$, the tgd $B \rightarrow A$ is called a *global-as-view projection* of Σ , denoted $r' \in \text{Gav}(\Sigma)$. Given a set of tgds Σ , we define the set \mathcal{A}_Σ of *affected* positions as the smallest set of positions such that, for all tgd $r \in \text{Gav}(\Sigma)$ of the form $r : B(X, Y) \rightarrow H(X, Z)$, we have:

- (i) $\forall v \in Y, \text{pos}(v, B) \subseteq \mathcal{A}_\Sigma$
- (ii) $\forall u \in X, (\text{pos}(u, H) \subseteq \mathcal{A}_\Sigma) \Rightarrow (\text{pos}(u, B) \subseteq \mathcal{A}_\Sigma)$

We say that Σ is *sticky* iff, for all tgd $r \in \text{Gav}(\Sigma)$ of the form above, and all $u \in X$ such that $\text{pos}(u, B) \subseteq \mathcal{A}_\Sigma$, the variable u occurs in only one atom of the body. This definition differs from [11] because of this last requirement “in only one atom”. In contrast, the definition from [11] requires that u occurs “only once” (in only one atom *and* in only one position).

THEOREM 5. *If Σ is a sticky set of tgds, then $\Sigma \downarrow$ is a set of lossless tgds, and therefore, for all set \mathcal{Q} of instances, there is a finite query rewriting for (Σ, \mathcal{Q}) .*

PROOF SKETCH. It can be checked that, for every sticky set Σ of tgds, and every atom A that can be simplified in a tgd $r \in \Sigma$, the set $\Sigma_{A,r}$ resulting from the simplification of (A, r) is also a sticky set of tgds. Assume now that $\Sigma \downarrow$ is sticky and $\Sigma \downarrow$ contains a tgd $r : B \rightarrow H$ which is not lossless. Since r is not lossless, there is an atom $A_h \in H$ and an atom $A_b \in B$ such that $\mathcal{V}_{A_b} \not\subseteq \mathcal{V}_{A_h}$. Observe that the tgd $B \rightarrow A_h$ belongs to $\text{Gav}(\Sigma \downarrow)$ and consider a variable $v \in \mathcal{V}_{A_b} \setminus \mathcal{V}_{A_h}$. By definition of $\mathcal{A}_{\Sigma \downarrow}$, this variable v occurs in an affected position, and the stickiness assumption ensures that v occurs only in the atom A_b , meaning that $v \notin \mathcal{V}_{B \setminus A_b}$. It follows that A_b can be simplified in r , and this contradicts the definition of $\Sigma \downarrow$. Therefore, every tgd in $\Sigma \downarrow$ is lossless. \square

We can observe that the (revisited) notion of stickiness is simultaneously a strict generalisation of: (1) the class of

lossless tgds; (2) the original notion defined in [11] from which it is inspired; and (3) the class of lav tgds, which was not yet covered by (2). Note also that stickiness could be combined with the (incomparable) notion of acyclicity discussed in Proposition 1 and/or the class of *sticky-join* tgds introduced in [11] to design an ever larger class of tractable settings. However, this is left as future work.

3.4 Integrating the Egds

This section provides a negative result on egds and query rewriting which will motivate two further contributions: (1) a novel technique, also in this section, that allows the *integrating* of some egds in a set of tgds; and (2) the study, in Section 4, of a richer notion of rewriting based on Datalog.

While the completeness result from Section 3.1 remains of clear interest with both tgds and egds, it turns out that the notion of query rewriting from Section 3.2 is in fact very limited under egds. Intuitively, this is because we considered a notion of *first-order* rewriting, while dealing with egds often requires the power of second-order (or, as we will see, the use of some integration technique which extends the schema). In fact, as illustrated by the following proposition, (Σ, \mathcal{Q}) is rarely rewritable under egds, even in the case where Σ consists of a single egd.

PROPOSITION 2. *There is no finite query rewriting for*

$$\Sigma = \{A(x, y), A(x, y') \rightarrow y = y'\}, \text{ and } \\ \mathcal{Q} = \{R(z, z)\}.$$

PROOF SKETCH. An infinite rewriting for (Σ, \mathcal{Q}) is the set of instances $\mathcal{R} = \{R_n\}_{n \geq 1}$ where each R_i is equal to $\{R(x_1, x_n)\} \cup \{A(x_i, y_i), A(x_{i+1}, y_i) \mid i \leq n-1\}$. We can check that this rewriting \mathcal{R} is not equivalent (up to \equiv) to any finite set of instances. Therefore, there is no finite rewriting for (Σ, \mathcal{Q}) . \square

Despite the above result, it has been observed in [22] that there are practical scenarios where egds can be ‘handled’ with a first order language. It is indeed possible, in some cases, to *integrate* these egds in the given set of tgds, as to compute a new set of tgds which, intuitively, does not interact with these egds. As a special case, this approach based on *integration*, covers the scenarios where the given sets of tgds and egds are already *non-conflicting*, as defined in [18] or [9]. However, we will also capture scenarios where the original set of tgds properly interacts with the egds.

As in [22] or [9], we next focus on *functional dependencies* rather than arbitrary egds. The reason for this is that the egds used in practice often consist of functional dependencies, and the functional dependencies have a more specific syntax which proves more convenient (in the context of integration). Recall that a functional dependency is a rule of the form $R_\alpha[K_\alpha] \rightarrow l_\alpha$ where R_α is a predicate of arity a_α , $K_\alpha \subseteq \{1, \dots, a_\alpha\}$ and $l_\alpha \in \{1, \dots, a_\alpha\} \setminus K_\alpha$. Given an instance M , we then let $M \models \alpha$ when, for all atoms of the form $R_\alpha(x_1, \dots, x_{a_\alpha})$ and $R_\alpha(x'_1, \dots, x'_{a_\alpha})$ in M , there either exists some $k \in K_\alpha$ such that $x'_k \neq x_k$, or it holds that $x'_{l_\alpha} = x_{l_\alpha}$. It is clear that a functional dependency can always be expressed by an equivalent egd. For example, for a binary predicate A , the dependency $\alpha : A[1] \rightarrow 2$ is equivalent to the egd $A(x, y), A(x, y') \rightarrow y = y'$.

Integration Heuristic. Given a set Σ of tgds and a functional dependency $\alpha : R_\alpha[K_\alpha] \rightarrow l_\alpha$, we let Σ_α be the set of tgds obtained as follows:

Start with $\Sigma_\alpha := \Sigma$
 Let D_α and F_α be two fresh predicates
 Let i_1, \dots, i_n be an ordering of K_α
 Add to Σ_α the two following tgds:
 $R_\alpha(x_1, \dots, x_{a_\alpha}) \rightarrow F_\alpha(x_{i_1}, \dots, x_{i_n}, x_{l_\alpha})$
 $D_\alpha(x_1, \dots, x_n) \rightarrow \exists y, F_\alpha(x_1, \dots, x_n, y)$
 For all tgd $r: B \rightarrow H$ in Σ
 For all atom $R_\alpha(t_1, \dots, t_{a_\alpha})$ in H
 (*) If $\{t_{i_1}, \dots, t_{i_n}\} \subseteq \mathcal{V}_B$ and $t_{l_\alpha} \notin \mathcal{V}_B$
 Add the atom $F_\alpha(t_{i_1}, \dots, t_{i_n}, t_{l_\alpha})$ to the body of r
 Add the tgd $B \rightarrow D_\alpha(t_{i_1}, \dots, t_{i_n})$ to Σ_α .

We say that α *interacts* with Σ when the lines below (*) in the above algorithm are actually used. That is, when there is a tgd $r: B \rightarrow H$ in Σ and an atom $A \in H$ of the form $R_\alpha(t_1, \dots, t_{a_\alpha})$ such that $\{t_i | i \in K_\alpha\} \subseteq \mathcal{V}_B$ and $t_{l_\alpha} \notin \mathcal{V}_B$. Note in particular that α does not interact with Σ when α is *non-conflicting* with Σ according to the definition of [9] (which would requires here that $\{t_i | i \in K_\alpha\} \not\subseteq \mathcal{V}_B$).

DEFINITION 3. We say that the integration of α succeeds in a set of tgds Σ iff the set of tgds Σ_α is such that:

- α does not interact with Σ_α , and
- for all tgd $B' \rightarrow H'$ in Σ_α , $B' \models F_\alpha[1, \dots, n] \rightarrow (n+1)$.

LEMMA 1. If the integration of α succeeds in Σ then, for all instances D and Q over the original schema such that $D \models \alpha$, the following statements are equivalent:

- $D \wedge \Sigma \wedge \alpha \models Q$
- $D \wedge \Sigma_\alpha \models Q$

PROOF SKETCH. Let $U \in \text{Fix}(D, \xrightarrow{\text{chase}}_{\Sigma_\alpha})$ and recall from Theorem 2 that U is a data rewriting for (D, Σ_α) . Under the assumptions that $D \models \alpha$ and α does not interact with Σ_α , we can check that $U \models \alpha$. It follows that U is also a data rewriting for $(D, \Sigma_\alpha \wedge \alpha)$. We can finally check that the following statements are all equivalent: $D \wedge \Sigma_\alpha \models Q$; $U \models Q$; $D \wedge \Sigma_\alpha \wedge \alpha \models Q$; and $D \wedge \Sigma \wedge \alpha \models Q$. \square

Consider now a set of functional dependencies \mathcal{F} and a set Σ of tgds. We say a set of tgd $\Sigma_{\mathcal{F}}$ of tgds *integrates* \mathcal{F} in Σ iff there exists a series $\alpha_1, \dots, \alpha_n \in \mathcal{F}$ and a series $\Sigma_0, \Sigma_1, \dots, \Sigma_n$ such that:

- $\Sigma_0 = \Sigma$, $\forall i \Sigma_{i+1} = (\Sigma_i)_{\alpha_i}$ and $\Sigma_n = \Sigma_{\mathcal{F}}$;
- for all i , the integration of α_i succeeds in Σ_i ; and
- there is no remaining $\alpha \in \mathcal{F}$ that interacts with $\Sigma_{\mathcal{F}}$.

We are now ready to formalize the property of interest which is ensured by the integration heuristic.

THEOREM 6. Given a set of tgds $\Sigma_{\mathcal{F}}$ that integrates a set of functional dependencies \mathcal{F} in a set of tgds Σ , for all instances D , all data rewriting $D_{\mathcal{F}}$ of (D, \mathcal{F}) , and all sets Q of instances, the following statements are equivalent:

- $D \wedge \Sigma \wedge \mathcal{F} \models Q$
- $D_{\mathcal{F}} \wedge \Sigma_{\mathcal{F}} \models Q$

PROOF SKETCH. The result can be proven by induction on the cardinality of \mathcal{F} using Lemma 1 and the result of separability which was established in [8]. \square

Note finally that, since \mathcal{F} is a set of functional dependencies, we can compute a data rewriting $D_{\mathcal{F}}$ for (D, \mathcal{F}) in polynomial time (data complexity) using any standard chase procedure. Combining Theorem 6 with the results of the previous section, we finally get the following result:

COROLLARY 1. Given $\Sigma_{\mathcal{F}}$ that integrates \mathcal{F} in Σ , if the set $\Sigma_{\mathcal{F}}$ is sticky, for all set Q of instances, the following problem is PTIME: given an instance D , does $D \wedge \Sigma \wedge \mathcal{F} \models Q$?

We finally provide an example of scenario taken from [22] which is covered by the approach described in this section:

$$\begin{aligned} \Sigma &= \{A(x, y) \rightarrow \exists z, B(x, z) \wedge C(z, y)\} \\ \mathcal{F} &= \{\alpha : B[1] \rightarrow 2\} \\ \Sigma_{\mathcal{F}} &= \left\{ \begin{array}{l} B(x, y) \rightarrow F_\alpha(x, y) \\ D(x) \rightarrow \exists y, F_\alpha(x, y) \\ A(x, y) \rightarrow D_\alpha(x) \\ A(x, y) \wedge F_\alpha(x, z) \rightarrow B(x, z) \wedge C(z, y) \end{array} \right\} \end{aligned}$$

3.5 Intermezzo: Constants and Free Variables

The goal of this section is to show how the previous results can be applied to more realistic *databases* (with constants and nulls) and *non-boolean queries* (with free variables).

3.5.1 Hard and Soft Constants

A database D is a set of atoms $R(t_1, \dots, t_n)$ where each term t_i is either a *variable* (also known as a *labelled null*) or a constant from a finite set $\Delta = \Delta_h \uplus \Delta_s$ where: Δ_h is a set of *hard constants* which are subject to the standard *unique name assumption* (UNA); and Δ_s is a set of *soft constants* which are not subject to the UNA (see e.g. [20]). Given a database D we denote by D^* the instance obtained from D as follows: (1) rename every $c \in \Delta$ into a variable v_c ; (2) for every $c \in \Delta$ introduce a fresh predicate R_c and add the atom $R_c(v_c)$; (3) introduce a fresh predicate $R_=$ and, for all $c, c' \in \Delta_h$ such that $c \neq c'$, add the atom $R_=(v_c, v_{c'})$. These definitions correspond to the standard encoding of constants and it can similarly be applied to boolean queries with constants. The following properties are then readily verified: (1) given a database D and a set Σ of tgds and egds, $D \wedge \Sigma$ is satisfiable iff $D^* \wedge \Sigma \not\models \{R_=(x, x)\}$; and (2) when $D \wedge \Sigma$ is satisfiable, for all set Q of instances, we have $D \wedge \Sigma \models Q$ iff $D^* \wedge \Sigma \models Q^*$. A less obvious observation, formalized below, is that this technique of simulation can also be used for query rewriting under integrable egds.

PROPOSITION 3.

- Given a set of tgds Σ , an integrable set of functional dependencies \mathcal{F} , and a database D , the formula $D \wedge \Sigma \wedge \mathcal{F}$ is satisfiable iff $D^* \wedge \Sigma_{\mathcal{F}} \not\models \{R_=(x, x)\}$.
- Under satisfiability, for all databases D , all data rewritings $D_{\mathcal{F}}$ of $(D^*, \Sigma_{\mathcal{F}})$ and all sets Q of instances, we have $D \wedge \Sigma \wedge \mathcal{F} \models Q$ iff $D_{\mathcal{F}} \wedge \Sigma_{\mathcal{F}} \models Q^*$.

COROLLARY 2. Given a set Σ of tgds and an integrable set \mathcal{F} of functional dependencies, if $\Sigma_{\mathcal{F}}$ is sticky, then, for all set Q of instances, the following problem is in PTIME: given a database D , does $D \wedge \Sigma \wedge \mathcal{F} \models Q$?

3.5.2 Free Variables and Equalities

Recall that a query Q is called a *union of conjunctive queries with equalities*, denoted $Q \in \text{UCQ}^=$, when Q is a first-order query of the form

$$Q = \{(x_1, \dots, x_a) \mid \bigvee_j Q_j\}$$

where each x_i is called a *free variable* and each clause Q_j is a finite conjunction of relational atoms and equality atoms (with constants, free variables, and existential variables). Given such a query Q , we may consider the set Q^* of instances obtained as follows: (1) rename every constant c into a variable v_c and add the atom $R_c(v_c)$ to each clause; (2) for every free variable x_i , introduce a predicate V_i and add the atom $V_i(x_i)$ to each clause; (3) introduce a predicate $R_=$ and replace every equality atom $(t = t')$ by the atom $R_=(t, t')$; and (4) let Q^* be the resulting set of clauses (which now consist of instances with variables only). Conversely, given an instance R^* of the extended schema (and for a fixed tuple (x_1, \dots, x_n) of free variable) we let R the set of relational and equality atoms over the original schema which is obtained by replacing every atom $R_c(u)$ by $(u = c)$ and every atom $V_i(u)$ by $u = x_i$. Given a set \mathcal{R}^* of instances over the extended schema, we finally denote by \mathcal{R} the $\text{UCQ}^=$ query of the form $\mathcal{R} = \{(x_1, \dots, x_a) \mid \bigvee \{R \mid R^* \in \mathcal{R}^*\}\}$. These definitions are exemplified below:

$$\begin{aligned} \Sigma &= \{A(u, v) \rightarrow B(u, u)\} \\ \mathcal{Q} &= \{(x_1, x_2) \mid B(x_1, x_2)\} \\ \mathcal{Q}^* &= \{B(x_1, x_2), V_1(x_1), V_2(x_2)\} \\ \mathcal{R}^* &= \mathcal{Q}^* \cup \{A(u, v), V_1(u), V_2(u)\} \\ \mathcal{R} &= \{(x_1, x_2) \mid B(x_1, x_2) \vee (\exists u, v, A(u, v) \wedge x_1 = u \wedge x_2 = u)\}. \end{aligned}$$

As next formalized, the above technique can be used to generalize the results of the previous sections to the case of non-boolean queries with constants and equalities:

PROPOSITION 4. *Given Σ , \mathcal{Q} , \mathcal{Q}^* , \mathcal{R}^* , \mathcal{R} as above where*

$$\mathcal{R}^* \in \text{Fix}(\mathcal{Q}^*, \xrightarrow{\text{resol}}_\Sigma)$$

the $\text{UCQ}^=$ query \mathcal{R} is a rewriting of the $\text{UCQ}^=$ query \mathcal{Q} . More formally, for all database D and all tuples \bar{c} of constants, the following statements are equivalent:

- \bar{c} is an answer of \mathcal{R} in D , denoted $\bar{c} \in \mathcal{R}(D)$
- \bar{c} is a certain answer of \mathcal{Q} in D under Σ , meaning that $\bar{c} \in \mathcal{Q}(D')$ for all instance D' such that $D \wedge \Sigma \models D'$.

COROLLARY 3. *We can use the technique of saturated resolution to compute a finite $\text{UCQ}^=$ rewriting for a given $\text{UCQ}^=$ query whenever there exists one (e.g. under stickiness).*

4. DATALOG REWRITING

As announced in the introduction, we now consider a more general notion of rewriting, called *Datalog rewriting*, which will prove to be a unifying paradigm of tractability for Datalog^\pm . More precisely, we will show that it captures the class of terminating dependencies (Section 4.1) and the class of guarded tgds (Section 4.2).

DEFINITION 4. *Given a set Σ of tgds and egds, and a set \mathcal{Q} of instances, a Datalog rewriting for (Σ, \mathcal{Q}) is a triple (σ_A, Γ, G) where*

- σ_A is a set of predicates which do not occur in Σ or \mathcal{Q} ;
- Γ is a finite set of tgds $B \rightarrow H$ where $\mathcal{V}_H \subseteq \mathcal{V}_B$;
- G is an instance of the form $G = \{G(\cdot)\}$ where $G \in \sigma_A$;
- for all σ_A -free instances D , it holds that:

$$D \wedge \Sigma \models Q \quad \text{iff} \quad D \wedge \Gamma \models G.$$

Note that, in the above definition, each tgd in Γ correspond to a standard Datalog rule (also known as a *full tgd*). Since Γ is required to be finite, Γ corresponds to a standard Datalog program. A predicate of σ_A will be called an *auxiliary* predicate and σ_A corresponds intuitively to an *intentional* schema. An σ_A -free instance is defined as an instance in which no predicate of σ_A occurs. That is, an σ_A -free instance corresponds intuitively to an *extensional* database. The instance G is finally known as the *goal* of the Datalog program (σ_A, Γ, G) and the predicate G , of arity 0, is known as the *goal predicate*. The following proposition finally summarizes the basic properties of Datalog rewritings:

PROPOSITION 5.

- *If a Datalog rewriting exists for (Σ, \mathcal{Q}) , the following problem is in PTIME: given D , does $D \wedge \Sigma \models \mathcal{Q}$?*
- *If there is a finite query rewriting for (Σ, \mathcal{Q}) then there is also a Datalog rewriting for (Σ, \mathcal{Q}) .*
- *There are some pairs (Σ, \mathcal{Q}) for which a Datalog rewriting exists while no finite query rewriting exists.*

PROOF SKETCH. The first point follows from the following observation: when Γ is a set of full tgds, we can compute a data rewriting \mathcal{U} for (D, Γ) in polynomial time (for a fixed Γ) using the chase, and we can then test in polynomial time (for a fixed instance G) whether $\mathcal{U} \models G$. For the second point, given a finite query rewriting \mathcal{R} for (Σ, \mathcal{Q}) and with letting $G = \{G(\cdot)\}$ for some fresh predicate G , we can observe that $(\{G\}, \{R \rightarrow G\}_{R \in \mathcal{R}}, G)$ is a Datalog rewriting for (Σ, \mathcal{Q}) . Finally, for $\Sigma = \{R(x, y), R(y, z) \rightarrow R(x, z)\}$ and $\mathcal{Q} = \{R(x, x)\}$, the pair $(\{G\}, \Sigma \cup \{R(x, x) \rightarrow G\}, G)$ is a Datalog rewriting of (Σ, \mathcal{Q}) while there is no finite (first-order) query rewriting for (Σ, \mathcal{Q}) . \square

4.1 From Termination To Datalog

This section revisits the criterion of *oblivious termination* which was introduced in [20] and presented in [10] as a language of the Datalog^\pm family. As discussed in [10], there are alternative criteria of termination that can be considered (see [26] for the current the state of the art). Note however that oblivious termination captures the case of *weakly acyclic* [15] sets of tgds and arbitrary sets of egds, and the results presented in this section can be extended to the classes discussed in [26]. In a nutshell, oblivious termination is based on (1) a technique of simulation that encodes the egds by means of tgds and (2) a standard notion of skolemization which generates a set of rules with function symbols (that is, a *logic program*). As observed in [20], this logic program enjoys a technical *bounded depth* property. In turn, we will show in this section that this bounded depth property ensures the existence of a Datalog rewriting.

Simulation. Given a set Σ of tgds and egds, we say that Σ' is a *substitution-free* simulation of Σ , denoted $\Sigma' \in$

$\text{Sim}_E(\Sigma)$, when Σ' is a set of tgds obtained from Σ using the simulation technique from [20] (which, unlike alternative techniques, avoid the use of substitution axioms). More precisely, we let $\Sigma' \in \text{Sim}_E(\Sigma)$ when E is a binary predicate which does not occur in Σ and Σ' can be computed with the following non-deterministic algorithm:

Start from $\Sigma' = \Sigma$.
 Add the following tgds to Σ' :
 $E(x, y) \rightarrow E(y, x)$
 $E(x, y), E(y, z) \rightarrow E(x, z)$
 For all predicates R occurring in Σ (of arity n)
 Add the following tgd to Σ' :
 $R(x_1, \dots, x_n) \rightarrow E(x_1, x_1), \dots, E(x_n, x_n)$
 Repeat (until fixed point)
 If there is a tgd $B \rightarrow H$ or an egd $B \rightarrow x=y$ in Σ' and a variable $x \in \mathcal{V}_B$ that occurs more than once in B
 Let x' be a fresh variable
 Replace one occurrence of x by x' in B
 Add the atom $E(x, x')$ to B
 For all egds $r : B \rightarrow x = y$ in Σ'
 Replace r by the tgd $B \rightarrow E(x, y)$
 Return Σ' .

Skolemization. Given a set of tgds Σ , we denote by P_Σ the logic program which is obtained by skolemizing Σ in a standard way. That is, for all $B(X, Y) \rightarrow H(X, Z)$ in Σ , the program P_Σ contains a rule $B(X, Y) \rightarrow H'(X)$ where H' is obtained from H by replacing every variable $z \in Z$ by a term $f(\bar{x})$ where f is fresh function symbol and the tuple \bar{x} is a fixed ordering of $X = \{x\}$. Given an instance I and such a logic program P_Σ we then denote by $P_\Sigma(I)$ the fixed point of I and P_Σ (also known as the minimal Herbrand model).

DEFINITION 5.

- We say that a set Σ of tgds ensures oblivious termination iff, for all finite instance I , $P_\Sigma(I)$ is finite.
- Given a set Σ of tgds and egds, we say that Σ ensures oblivious termination iff there exists $\Sigma' \in \text{Sim}_E(\Sigma)$ such that Σ' ensures oblivious termination.

Bounded Depth. Given a skolem term t with variables and function symbols, we define the *depth* $d(t)$ of t in a standard way. More precisely, given a variable x we let $d(x) = 1$, and given a term $t = f(t_1, \dots, t_n)$, we let $d(t) = 1 + \max_{i \leq n} (d(t_i))$. Given a set of tgds Σ , an instance D and an integer k we denote by $P_\Sigma^k(D)$ the set of atoms with skolem terms which is obtained by applying (inductively) all the rules $B(X, Y) \rightarrow H'(X)$ of P_Σ , but only for the valuations θ of $X \cup Y$ such that, for all $u \in X \cup Y$, the depth of $\theta(u)$ is at most k . It can be checked that $P_\Sigma^k(I)$ is finite and well-defined whenever Σ , I and k are finite. In particular, the order of application of the rules does not matter. Note here that $P_\Sigma^k(I)$ is a skolem instance (with skolem terms) rather than a standard instance (with variables only) but he definitions from Section 3.1 can nonetheless be extended in a natural way. In particular, given a set \mathcal{Q} of instances, we let $P_\Sigma^k(I) \models \mathcal{Q}$ iff, for all $Q \in \mathcal{Q}$, there is a mapping θ from \mathcal{V}_Q to the terms of $P_\Sigma^k(I)$ such that $Q[\theta] \subseteq P_\Sigma^k(I)$.

DEFINITION 6. Given a set Σ of tgds and a set \mathcal{Q} of instances, we say that (Σ, \mathcal{Q}) has bounded depth iff there exists a finite integer k (depending only on (Σ, \mathcal{Q})) such that, for all instances D , the following statements are equivalent:

- $D \wedge \Sigma \models \mathcal{Q}$
- $P_\Sigma^k(D) \models \mathcal{Q}$

The following result was established in [20]:

LEMMA 2. If Σ is a finite set of tgds ensuring oblivious termination, there exists k (depending only on Σ) such that, for all instances D , $P_\Sigma(D) = P_\Sigma^k(D)$. Therefore, for all sets \mathcal{Q} of instances, (Σ, \mathcal{Q}) has bounded depth.

We next present the main result of this section.

THEOREM 7. Given a set Σ of tgds and a set \mathcal{Q} of instances, if (Σ, \mathcal{Q}) has bounded depth (and if we know the bound k), we can compute a Datalog rewriting for (Σ, \mathcal{Q}) .

PROOF SKETCH. The key idea of the proof is to use fresh predicate symbols to simulate the effect of the function symbols of P_Σ^k . Intuitively, every atom $R(\bar{t})$ with skolem terms can indeed be simulated with a standard atom $R_s(\bar{x})$, with variables only, where s encodes the “shape” of each term, while \bar{x} corresponds to the variables that occur in \bar{t} . For instance:

$$R(x, f\langle x, y \rangle, g\langle y, f\langle x, z \rangle \rangle) \approx R_{1, f\langle 1, 2 \rangle, g\langle 2, f\langle 1, 3 \rangle \rangle}(x, y, z)$$

For a fixed bound k , for every rule $B(X, Y) \rightarrow H'(X)$ in P_Σ and for every valuation θ of $X \cup Y$ such that $\max\{d(\theta(v)) \mid v \in X \cup Y\} \leq k$ we can then translate every skolem atom in the rule $B[\theta] \rightarrow H[\theta]$ into a standard atom. For instance, if $k = 2$ and P_Σ contains only one binary skolem function f , we can replace the rule

$$A(x, y) \rightarrow B(x, y, f(x, y))$$

by a set of Datalog rules containing, among others, the rules:

$$\begin{aligned} A(x, y) &\rightarrow B_{1, 2, f\langle 1, 2 \rangle}(x, y) \\ A_{1, f\langle 1, 1 \rangle}(x) &\rightarrow B_{1, f\langle 1, 1 \rangle, f\langle 1, f\langle 1, 1 \rangle \rangle}(x) \\ A_{1, f\langle 1, 2 \rangle}(x, y) &\rightarrow B_{1, f\langle 1, 2 \rangle, f\langle 1, f\langle 1, 2 \rangle \rangle}(x, y) \\ A_{f\langle 1, 2 \rangle, f\langle 1, 3 \rangle}(x, y, z) &\rightarrow B_{f\langle 1, 2 \rangle, f\langle 1, 3 \rangle, f\langle f\langle 1, 2 \rangle, f\langle 1, 3 \rangle \rangle}(x, y) \\ &\dots \end{aligned}$$

Consider now \mathcal{Q} of the form $\mathcal{Q} = \{G()\}$ such that (Σ, \mathcal{Q}) has bounded depth k . Let σ_A^k be the set predicates R_s where s encodes a shape of depth $\leq k$. Let Γ^k be set of Datalog rules resulting from the above construction. We can check in this case that $(\sigma_A, \Sigma^k, \{G()\})$ is a Datalog rewriting for (Σ, \mathcal{Q}) . In the general case, (when \mathcal{Q} is not already of the form $\{G()\}$), we may introduce a fresh 0-ary predicate G , consider all the tgds $r_Q : Q \rightarrow G()$ where $Q \in \mathcal{Q}$, and consider all the valuations θ of X_Q such that $\max\{d(\theta(x)) \mid x \in X_Q\} \leq k$. We can then encode each of these rules with a Datalog rule over $\sigma_A^k \cup \{G\}$, and we can conclude as in the previous case. \square

COROLLARY 4. There is an algorithm that, given a set Σ of tgds and egds ensuring oblivious termination, and given a set \mathcal{Q} of instances, computes a Datalog rewriting for (Σ, \mathcal{Q}) .

4.2 From Guardedness To Datalog

In this section, we consider the class of *guarded* tgds. Intuitively, we will say that a tgd is guarded when there is an atom in the body (called a *guard*) that contains all the *universal* variables. In turn, a variable is called universal iff it occurs both in the body and the head. Note that the variables that occur only in the body are not taken into account in this definition of guardedness. Therefore, the class

of guarded tgds contains, as a special case, the tgds $B \rightarrow G()$ where B is an arbitrary instance and G is a 0-ary predicate (for example, a goal predicate) because such tgds have no universal variable. Another example of guarded tgd is

$$A(x, y, z), B(i, x, y), B(j, y, z) \rightarrow \exists k, B(k, x, z)$$

where the set of universal variables is $\{x, z\}$ and the atom $A(x, y, z)$ is a guard. In contrast, the tgd

$$B(i, x, y), B(j, y, z) \rightarrow \exists k, B(k, x, z)$$

is *not* guarded since there is no atom in the body that contains both x and z .

DEFINITION 7. A tgd of $B \rightarrow H$ is guarded iff there is a atom $G \in B$ such that $(\mathcal{V}_B \cap \mathcal{V}_H) \subseteq \mathcal{V}_G$.

4.2.1 The Case of β -Guardedness

A special class of guarded tgds was considered in [8, 10], called β -guarded tgds in this section, that complies with the following definition:

DEFINITION 8. A tgd $B \rightarrow H$ is called β -guarded iff there exists an atom $G \in B$ such $\mathcal{V}_B \subseteq \mathcal{V}_G$.

Note that a β -guarded tgd is (only) a special case of guarded tgds since the requirement $\mathcal{V}_B \subseteq \mathcal{V}_G$ is stronger than $(\mathcal{V}_B \cap \mathcal{V}_H) \subseteq \mathcal{V}_G$. For example, the tgd

$$A(x, y, z), B(i, x, y), B(j, y, z) \rightarrow \exists k, B(k, x, z)$$

is guarded but not β -guarded. While β -guardedness is slightly less general, it was shown in [8] that the class of β -guarded tgds remains a very natural class to consider. In particular, it was shown in [8] that we only need β -guardedness (as opposed to general guardedness) to cover interesting classes of ontologies, including languages from the DL-lite family [3]. Another important advantage of β -guardedness is that it ensures an useful property, established in [8], called the *bounded guard-depth property*. This property (defined for β -guarded tgds only) proves indeed very relevant here as it coincides in fact with the general property of *bounded depth* which was discussed in the previous section. Combining the results in [8] with this observation, we obtain the following results:

LEMMA 3. If Σ is a finite set of β -guarded tgds and \mathcal{Q} is a finite set of instance, then (Σ, \mathcal{Q}) has bounded depth k for some computable k that depends both on Σ and \mathcal{Q} .

COROLLARY 5 (OF THEOREM 7). There is an algorithm that, given a set Σ of β -guarded tgds and a set \mathcal{Q} of instances, computes a Datalog rewriting for (Σ, \mathcal{Q}) .

As already discussed, a β -guarded tgd is only a special case of guarded tgd, and the work of [5] suggests that there is no trivial reduction from the class of guarded tgds to the class of β -guarded tgds. This is why we consider an alternative approach, in the following section, for the more general case.

4.2.2 From Guardedness To Flatness

In this section, we provide a proof (sketch), based on a technical notion of *flatness*, for the following result:

THEOREM 8. For every finite set Σ of guarded tgds and set \mathcal{Q} of instances, there is a Datalog rewriting for (Σ, \mathcal{Q}) .

The key idea of the proof can be summarized as follows: when Σ is a set of guarded tgds, there exists an equivalent set of tgds Σ' which enjoys the *flat chase property*. This property meaning intuitively that it is sufficient to chase the tgds $B(X, Y) \rightarrow H(X, Z)$ of Σ' for the renamings of X that map X to the variables of the original instance D (which intuitively correspond to constant values). In turn, the flat chase property can be linked with the property of bounded depth (for the depth $k = 1$) and Theorem 7 can be used again to prove the existence of a Datalog rewriting.

Flat Chase. Consider a triple (D, I, J) of instances and a set Σ of tgds. When $I \xrightarrow{\text{chase}}_{\Sigma} J$ we say that the chase step is *flat* with respect to D , denoted $I \xrightarrow{D}_{\Sigma} J$, when there is a tgd $B(X, Y) \rightarrow H(X, Z)$ in Σ and two substitutions θ_X and θ_Y such that:

- $B[\theta_X, \theta_Y] \subseteq I$ and $J = I \cup H[\theta_X, \theta_Z]$ for some I -fresh renaming θ_Z ; and
- in addition, for all $x \in X$, it holds that $\theta_X(x) \in \mathcal{V}_D$.

Given an instance U , we finally let $U \in \text{Flat}(D, \Sigma)$ when there exists a finite derivation of the form

$$D = I_1 \xrightarrow{D}_{\Sigma} I_2 \xrightarrow{D}_{\Sigma} \dots \xrightarrow{D}_{\Sigma} I_n = U.$$

DEFINITION 9. Given a set Σ of tgds and a set \mathcal{Q} of instances, we say that (Σ, \mathcal{Q}) has the flat chase property iff, for all instance D the following statements are equivalent:

- $D \wedge \Sigma \models \mathcal{Q}$
- $\exists U \in \text{Flat}(D, \Sigma), U \models \mathcal{Q}$

LEMMA 4. If (Σ, \mathcal{Q}) has the flat chase property, then (Σ, \mathcal{Q}) has bounded depth (for the depth $k = 1$) and therefore, there exists a Datalog rewriting for (Σ, \mathcal{Q}) .

The proof of Theorem 8 finally relies on Lemma 5 below.

LEMMA 5. For all finite set Σ of guarded tgds and all finite set \mathcal{Q} of instances, there exists a finite set Σ' of tgds such that $\Sigma \equiv \Sigma'$ and (Σ', \mathcal{Q}) has the flat chase property.

PROOF SKETCH. Given a tgd $r : B(X, Y) \rightarrow H(X, Z)$, a refinement of r is a tgd $r' = B' \rightarrow H'$ where $B' = B[\theta_X, \theta_Y]$ and $H[\theta_X, \theta_Z]$ for some renamings $\theta_X, \theta_Y, \theta_Z$ where θ_Z is a B' -fresh renaming of Z . We say that r' is a *careful* refinement of r when, in addition, θ_Y is a H' -fresh renaming. Given a set of tgds Σ , we let $\text{Ref}(\Sigma)$ (resp. $\text{CRef}(\Sigma)$) the sets of tgds corresponding to a refinement (resp. careful refinement) of some tgd of Σ . Given a tgd $r : B \rightarrow H$ we say that r is of the *split form*

$$r : G(X, U), B'(X', U', V) \rightarrow H(X, Z)$$

when X and Z are defined as usual, G is a subset of B satisfying $X \subseteq \mathcal{V}_G$, U is the set of remaining variables in G , B' is the set of atoms in $B \setminus G$, V is the set of variables that occur only in B' , and finally $(X', Y') = (X \cap \mathcal{V}_{B'}, Y \cap \mathcal{V}_{B'})$.

Given two sets Σ_1 and Σ_2 of tgds, we say that a tgd r_3 is derived from (Σ_1, Σ_2) when there exists a tgd $r_1 \in \text{CRef}(\Sigma_1)$ and a tgd $r_2 \in \text{Ref}(\Sigma_2)$ such that:

- $r_1 : B_1(X_1, Y_1) \rightarrow H_1(X_1, Z_1)$;

- $r_2 : G_2(X_2, U_2), B'_2(X'_2, U'_2, V_2) \rightarrow H_2(X_2, Z_2)$
 where : 1. G_2 is non-empty and contained in H_1
 2. $(X'_2 \cup U'_2 \cup V_2)$ is disjoint from $(Z_1 \cup Y_1)$
 3. Z_2 is disjoint from $(X_1 \cup Y_1 \cup Z_1)$; and
- $r_3 = (B_1 \cup B'_2) \rightarrow (H_1 \cup H_2)$.

It follows here from 1. and 2. that $X'_2 \cup U'_2 \subseteq X_1$ while V_2 is disjoint Y_1 . Therefore, the tgd r_3 is of the slit form

$$r_3 : G_3(X_3, U_3), B'_3(X'_3, \emptyset, V_3) \rightarrow H_3(X_3, Z_3)$$

where $G_3 = B_1$, $X_3 = X_1$, $U_3 = Y_1$, $B'_3 = B'_2$, $X'_3 = X'_2 \cup U'_2$, $V_3 = V_2$, $H_3 = H_1 \cup H_2$, and $Z_3 = Z_1 \cup Z_2$.

Fact 1. When a tgd r_3 is derived from (Σ_1, Σ_2) , it holds that $\Sigma_1 \cup \Sigma_2 \models r_3$

Proof Sketch. Using the previous notations, we have

$$(B_1 \cup B'_2) \xrightarrow{\text{chase}_{\{r_1\}}} (B_1 \cup B'_2 \cup H_1) \xrightarrow{\text{chase}_{\{r_2\}}} (B_1 \cup B'_2 \cup H_1 \cup H_2)$$

and it follows that $\{r_1, r_2\} \models r_3$.

Given an integer k we say that a tgd r is k -guarded if r is of the form

$$r : G(X, U), B'(X', U', V) \rightarrow H(X, Z)$$

for some instance G with only one atom (that is, a guard atom), and there exists a set of instances $(B^i)_{i \leq n}$ called a (G, k) -decomposition of B such that: (1) $B' = \cup_{i \leq n} B^i$, (2) for each $i \leq n$, B^i has at most $(k-1)$ atoms; and (3) for all $i \neq j$, it holds that $\mathcal{V}_{B^i} \cap \mathcal{V}_{B^j} \subseteq \mathcal{V}_G$. We define the guard width of a guarded tgd r , denoted $\text{gw}(r)$ as the smallest k such that r is k -guarded. In contrast, we define the left width of a tgd r , denoted $\text{lw}(r)$, as the number of atom in the body of r . Note that, every guarded tgd is such that $\text{gw}(r) \leq \text{lw}(r)$. Given a set of tgds Σ , we finally let $\text{gw}(\Sigma) = \max\{\text{gw}(r), r \in \Sigma\}$ and $\text{lw}(\Sigma) = \max\{\text{lw}(r), r \in \Sigma\}$.

Fact 2. When a tgd r_3 is derived from (Σ_1, Σ_2) , it holds that $\text{gw}(r_3) \leq \max(\text{gw}(\Sigma_1), \text{lw}(\Sigma_2))$.

Proof Sketch. It can be checked that $\text{lw}(r') \leq \text{lw}(r)$ whenever r' is a refinement of r and $\text{gw}(r') \leq \text{gw}(r)$ whenever r' is a careful refinement of r . Let $r_1 \in \text{CRef}(\Sigma_1)$ be a k -guarded tgd of form

$$r_1 : G_1(X_1, U_1), B'_1(X'_1, U'_1, V_1) \rightarrow H_1(X_1, Z_1)$$

and a (G_1, k) -decomposition $B'_1 = \cup_{i \leq n} B^i_1$. Let $r_2 \in \text{Ref}(\Sigma_2)$ such that $\text{lw}(r_2) \leq k$ and r_2 is of the form

$$r_2 : G_2(X_2, U_2), B'_2(X'_2, U'_2, V_2) \rightarrow H_2(X_2, Z_2).$$

Finally, let r_3 be the tgd derived from r_1 and r_2 , and recall that r_3 is of the form

$$r_3 : G_3(X_3, U_3), B'_3(X'_3, \emptyset, V_3) \rightarrow H_3(X_3, Z_3)$$

where $G_3 = G_1 \cup B'$, $X_3 = X_1$, $U_3 = U_1 \cup V_1$, and $B'_3 = B'_2$. Since $\text{lw}(r_2) \leq k$ and G_2 is non-empty, we have $|B'_2| \leq (k-1)$. For all $i \leq n$, we have $\mathcal{V}_{B^i_1} \cap \mathcal{V}_{B^i_2} \subseteq X_1 \subseteq \mathcal{V}_{G_1}$. Therefore, the set of instances $\{B^i_1\}_{i \leq n} \cup \{B^i_2\}$ is a (G_1, k) decomposition of the body of r_3 and r_3 is k -guarded.

Given a set of tgds Σ , we define the flatening of Σ , denoted Σ^∞ , as the minimal set of tgds Σ' such that $\Sigma \subseteq \Sigma'$ and Σ' contains all the tgds that can be derived from (Σ', Σ) .

Fact 3. For all set Σ of tgds and all set \mathcal{Q} of instances, $(\Sigma^\infty, \mathcal{Q})$ has the flat chase property.

Proof Sketch. Consider an instance D and suppose that $D \wedge \Sigma \models \mathcal{Q}$. By completeness of the chase, we know that there exists a derivation

$$D = I_0 \xrightarrow{\text{chase}_\Sigma} I_1 \xrightarrow{\text{chase}_\Sigma} \dots \xrightarrow{\text{chase}_\Sigma} I_n$$

such that $I_n \models \mathcal{Q}$. If this derivation is flat, and since $\Sigma \subseteq \Sigma^\infty$, we have $I_n \in \text{Flat}(D, \Sigma^\infty)$ and $I_n \models \mathcal{Q}$. Suppose now that the derivation is not flat and consider the first integer $j \leq n$ such that I_j is obtained from I_{j-1} with a chase step which is not flat. We can check that there exists a tgd $r_2 : B_2(X_2, Y_2) \rightarrow H_2(X_2, Z_2)$ in $\text{Ref}(\Sigma_2)$ such that $B_2 \subseteq I_{j-1}$, $I_j = I_{j-1} \cup H_2$ and $X_2 \not\subseteq \mathcal{V}_D$. Consider a guard atom $G(\bar{v})$ for r_2 . Since $X \subseteq \{\bar{v}\}$ and $X \not\subseteq \mathcal{V}(D)$, there exists some $i \leq j$ such that $G(\bar{v}) \in (I_i \setminus I_{i-1})$. Let G_2 be the set of all the atoms of B_2 that belong to $(I_j \setminus I_{j-1})$. Write r_2 under the form

$$r_2 : G_2(X_2, U_2), B'_2(X'_2, U'_2, V_2) \rightarrow H_2(X_2, Z_2).$$

and observe that G_2 is non-empty (since it contains $G(\bar{v})$). Considering $N_j = \mathcal{V}_{I_j \setminus I_{j-1}}$, since the derivation is flat from the step 1 to the step j , we can observe that the only atom of I_{i-1} containing a variable of N_j are the atoms of $I_j \setminus I_{j-1}$. Therefore, $(U'_2 \cup V_2)$ is disjoint from N_j . By definition of careful refinements, we can now consider a tgd $r_1 \in \text{CRef}(\Sigma)$ of the form

$$r_1 : B_1(X_1, Y_1) \rightarrow H_1(X_1, Z_1)$$

where Y_1 is set of variables disjoint from (X'_2, U'_2, V_2) , $B_1[\theta_{Y_1}] \subseteq I_{i-1}$ for some renaming θ_{Y_1} of Y_1 and $I_i = I_{i-1} \cup H_1$. Let $r_3 = B_3 \rightarrow H_3$ be the tgd such that $B_3 = B_1 \cup B'_2$ and $H_3 = H_1 \cup H_2$, and observe that $r_3 \in \Sigma^\infty$. Let θ' be an injective renaming of $Z_1 \cup Z_2$ into a set of fresh variables (disjoint from \mathcal{V}_{I_n}) and for all $k \leq j$, let $I'_k = I_k[\theta']$. We can observe that we have

$$D = I_0 \xrightarrow{\text{chase}_\Sigma} \dots \xrightarrow{\text{chase}_\Sigma} I_{j-1} \xrightarrow{\text{chase}_{r_3}} I'_j \xrightarrow{\text{chase}_\Sigma} \dots \xrightarrow{\text{chase}_\Sigma} I'_n$$

where the step $I_{j-1} \xrightarrow{\text{chase}_{r_3}} I'_j$ is now a flat step.

We can finally generalize the above construction to show (by induction on n) that every chase derivation of D by Σ (leading to $I_n \models \mathcal{Q}$) can be transformed into a flat derivation of D by Σ^∞ (leading to $I'_n \models \mathcal{Q}$)

Given a tgd $r : B(X, Y) \rightarrow H(X, Z)$, we define a left-core projection of r as a minimal (for \subseteq) set of atoms $B' \subseteq B$ such that $B' = B[\theta_Y]$ for some renaming θ_Y of Y . Given a set of tgds Σ , we let $LC(\Sigma)$ be the set of instances B' such that B' is a left-core projection of some tgd $r \in \Sigma$.

Fact 4. Given a finite schema σ_0 , an integer k_0 , and infinite set Σ of k_0 -guarded tgds over σ_0 , the set $LC(\Sigma)$ is finite up to isomorphism.

Proof Sketch. On a fixed schema σ_0 , there are (up to isomorphism) only a finite number of atoms that can be used as a guard G for a decomposition $(B^i)_{i \leq n}$. For a fixed integer k_0 and for every instance $B = G \cup \bigcup_i B^i$ corresponding to the left-core projection of some tgd, there is only a finite number of possible blocks B^i satisfying $|B^i| \leq k-1$ that can be used in the composition, up to bijective renaming of $\mathcal{V}_{B_i} \setminus G$. It follows that each $B \in LC(\Sigma)$ is finite and that $LC(\Sigma)$ is finite up to isomorphism.

Given a set of tgds Σ and an integer k , we say that a tgd $r : B \rightarrow H$ is a k -tgd of Σ , denoted $\Sigma^{(k)}$ when there exists a tgd $B' \rightarrow H'$ in the flattening Σ^∞ of Σ such that: (1) H is a subset of H' containing at most k atoms; and (2) B is a left-core projection of $B' \rightarrow H'$. As a corollary of Fact 4, it can be observed that $\Sigma^{(k)}$ is finite (up to isomorphism) whenever Σ is a finite set of guarded tgds.

Fact 5. For all sets Σ of guarded tgds, all sets \mathcal{Q} of instances and all integer k such that

$$k \geq \max\{|B|, B \in LC(\Sigma^\infty) \cup \mathcal{Q}\}$$

$(\Sigma^{(k)}, \mathcal{Q})$ enjoys the flat chase property.

Proof Sketch. The property can be shown by adapting the proof of Fact 3. Indeed, a flat derivation by Σ^∞ can be modified in two ways that preserves completeness: (1) one may replace the body of a tgd by a left-core of this tgd; and (2) one may select, for each tgd $B \rightarrow H$, the atoms from H which will be used later in the derivation (as to enforce that $|H| \leq k$).

We can finally use Fact 5 to conclude the proof of Lemma 5 and Theorem 8. \square

5. CONCLUSION AND FUTURE WORK

This paper presented contributions along two axes:

More General Classes. Considering a complete resolution procedure extends the possible fields of applications of Datalog $^\pm$. Resolution can be used, first, to generalize existing tractability criteria (such as stickiness) but it can also be used, in practice, without any syntactic assumption. A technique of integration or simulation however proves often useful (and sometimes necessary) to handle egds. Datalog rewriting finally covers the three main paradigms of Datalog $^\pm$: stickiness, termination and guardedness. There are also alternative paradigms of tractability that were considered in [4] which have not been discussed here (to simplify the discussion) but are yet to compare with the class of Datalog-rewritable dependencies. In particular, a natural question (left as future work) is the following: is Datalog rewriting always possible under *bounded-tree width*[13]?

More Efficient Algorithms. This paper introduced several algorithms and heuristics that can be of clear practical use. In particular, the resolution can certainly (at least, in some contexts) be more efficient than the chase. The technique of Datalog Rewriting can also prove useful in practice. In particular, one may consider Magic Set or any other standard optimisation of Datalog to improve efficiency. Nonetheless, there is still a long road ahead because the algorithms of Datalog Rewriting proposed in Section 4 are (for the moment) very much non-optimal. An interesting and challenging question that remains is the following: how to compute *in practice* a Datalog rewriting of reasonable size?

6. REFERENCES

- [1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [2] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *J. Comput. Syst. Sci.*, 43(1):62–124, 1991.
- [3] A. Acciarri, D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QuOnto: Querying ontologies. In *AAAI*, pages 1670–1671, 2005.
- [4] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. Extending decidable cases for rules with existential variables. In *IJCAI*, pages 677–682, 2009.
- [5] V. Barany, G. Gottlob, and M. Otto. Querying the guarded fragment. In *LICS*, 2010.
- [6] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *ICALP*, pages 73–85, 1981.
- [7] F. Bry, T. Furche, B. Marnette, C. Ley, B. Linse, and O. Poppe. SPARQLLog: SPARQL with rules and quantification. In *Semantic Web Information Management: A Model-Based Perspective*, pages 341 – 369. Springer, 2010.
- [8] A. Cali, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *KR*, pages 70–80, 2008.
- [9] A. Cali, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. In *PODS*, pages 77–86, 2009.
- [10] A. Cali, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS*, pages 228–242, 2010.
- [11] A. Cali, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in datalog+/- . In *RR*, pages 1–17, 2010.
- [12] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.
- [13] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
- [14] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.
- [15] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
- [16] E. Grädel. Decision procedures for guarded logics. In *CADE*, pages 31–51, 1999.
- [17] J. Hladik. Implementation and optimisation of a tableau algorithm for the guarded fragment. In *TABLEAUX*, pages 145–159, 2002.
- [18] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.
- [19] Y. Kazakov and H. de Nivelle. A resolution decision procedure for the guarded fragment with transitive guards. In *IJCAR*, volume 3097 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2004.
- [20] B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.
- [21] B. Marnette. *Tractable Schema Mappings Under Oblivious Termination*. PhD thesis, Oxford University, 2010.
- [22] B. Marnette, G. Mecca, and P. Papotti. Scalable data exchange with functional dependencies. *PVLDB*, 3(1):105–116, 2010.
- [23] A. Nash, P. A. Bernstein, and S. Melnik. Composition of mappings given by embedded dependencies. In *PODS*, pages 172–183, 2005.
- [24] H. D. Nivelle and M. D. Rijke. Deciding the guarded fragments by resolution. *Journal of Symbolic Computation*, 35:21–58, 2003.
- [25] H. Pérez-Urbina, B. Motik, and I. Horrocks. Rewriting Conjunctive Queries over Description Logic Knowledge Bases. In *Proc. of the Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008)*, Nantes, France, March 2008. Springer.
- [26] F. Spezzano and S. Greco. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.